Similar but Foreign: Link Recommendation Across Communities

Yao Ge[1], Chunyao Song[1], Tingjian Ge[2]

[1] *Tianjin Key Laboratory of Network and Data Science Technology*
College of Computer Science, Nankai University
[2] *University of Massachusetts Lowell*

Abstract

Complex networks have been used widely to model complex systems composed of interacting entities. It is possible to recommend new relationships between entities according to network topology and the entities' properties. The likelihood of forming a missing or potential relationship is often indicated by similarity. As community clusters are also based on entities' similarities, traditional link recommendation/prediction methods naturally tend to recommend links within a community. However, potential links valuable across communities are often overlooked. We focus on link recommendation across communities based on homogeneous and heterogeneous information networks, which aims to improve the diveristy of recommender systems. We propose a solution to this problem through novel similarity calculation and heterogeneous network embedding methods. Our comprehensive experiments over real-world datasets and synthetic datasets show that our methods strike a good balance between accuracy and efficiency, while generating valuable unconventional recommendations in practical application scenarios.

Keywords: Link Recommendation; Inter-Community; Random Walk; Limited-Hop; Network Embedding

1. Introduction

A typical network consists of nodes and edges, where nodes represent entities in real systems and edges express the relationships between entities. Many complex systems can be modeled in the form of networks. Recommending or predicting unseen relationships between entities can reveal useful information in network analysis. Link prediction is applicable in many areas, such as the prediction of unknown interactions between proteins [1], friend recommendation [2], and product recommendation [3]. One of the main uses of link prediction is for link/relationship recommendation. Over the years, multiple approaches have been proposed to solve this problem. Traditional solutions include neighbor-based, path-based, probability-based, matrix-factorization based, and classifier-based methods [4]. Recently, with the development of network representation learning, network embedding [5] is widely used for link prediction, especially on the heterogeneous information networks.

Similar entity recommendation is one of the main application scenarios of link recommendation. Current link recommendation methods dominantly recommend links between pairs of nodes that are within a certain range in graph topology, which is typically within the same community. For example, friend recommendations typically always recommend people in the same organization due to the community structure present in the data, and product recommendations typically always show the ones in the same cluster (as the one already bought) that tend to be co-purchased together. An obvious drawback is that current recommendation systems usually recommend entities within the same community, which limits the potential of recommending a wider range of entities. However, inter-community links, when they exist, can be very valuable for recommendation, because they often provide values in terms of diveristy and information propagation [6]. In the study of sociologist Granovetter back in 1973, such long range links involving *weak ties* are indispensable to individuals' opportunities and to their intergration into communities [6]. By contrast, they find that structurally close edges are heavily redundant in terms of information access. Therefore, the inter-community links can provide possible expansions for link recommendation. Moreover, link recommendation across communities will greatly increase the diversity of recommendations.

Heterogeneous information networks (HeINs) contain multiple types of nodes and interactions. For clarity, we call the general networks whose nodes belong to the same type as homogeneous information networks (HoINs). Due to the complexity of HeINs, let us first clarify the concept of community there. Most community detection studies on HeINs focus on the generation of communities based on a certain type of nodes. For example, if we consider the co-purchase network, it is not reasonable that users and products belong to the same community. Analogously, for HeINs, although we take full advantage of the heterogeneous information network, when we perfor-

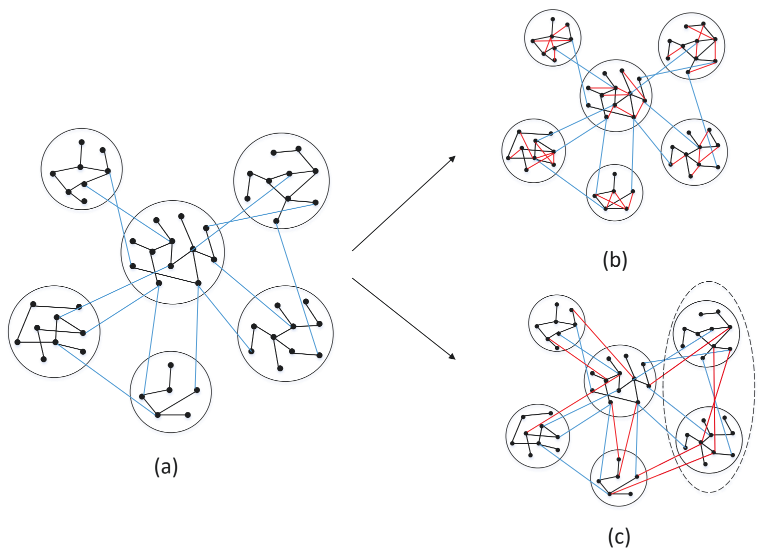


Figure 1: Link recommendation across communities.

m recommendation, we specify a target type in advance (e.g., we perform inter-community recommendations on products only on co-purchase network which includes two types of nodes). Next, let us take Figure 1 as an example to illustrate link recommendation across communities. Figure 1(a) is the original graph, where nodes are organized into six communities (each black oval represents a community). For HeINs, it only shows the nodes that belong to the target type, and other types of nodes are omitted. Figure 1(b) represents the network following the traditional link recommendation, where the connections of nodes within the original community become tighter, but recommendations hardly reach nodes in other communities. Figure 1(c) illustrates the network following our recommendation, where potentially probable connections are found between entities far enough in different communities, improving the diversity of recommendations. The newly discovered missing links may alter the true community structure as well, as illustrated in the dashed oval in Figure 1(c). Let us look at some more concrete examples.

Example 1. *A recommender system may recommend to a researcher, as potential collaborators, only researchers in the same field with higher topology similarities. However, significant research results today increasingly require interdisciplinary collaboration. For example, a data scientist majoring in computer science may contribute to gene database and protein-protein in-*

teraction prediction in biology. The collaborations across fields will promote the mutual development of multiple disciplines, and the overall scientific advancement.

Example 2. *A similar situation arises in product recommendations. There are multiple E-commerce platforms, such as Amazon, Ebay, and Walmart. Suppose a customer has just purchased several shirts. A typical recommender system may recommend to him/her some other shirts based on a co-purchase graph. However, such recommendations will only be useful in a very short period of time before the customer has bought enough shirts. In contrast, an across-community recommendation, e.g., recommending shoes, which are not immediately linked but have intrinsic connections, may suit the need of the customer better.*

As suggested in the examples, we propose solutions for both HoINs and HeINs, shifting towards the inter-community part of link recommendation, while keeping the efficiency and accuracy at the same time. In order to solve the problem, a naive solution is to perform community detection first, then filter the results of link prediction based on the communities. However, community detection methods themselves involve various quality and accuracy metrics [7]. It is hard to guarantee that community detection algorithms will provide sufficient results to satisfy our needs of inter-community link recommendation, and the two-phase procedure will lead to a high time complexity.

First, we propose a novel algorithm of link recommendation across communities in HoINs. Although we want to calculate the similarities between pairs of nodes that are relatively far apart, there is no need to compute the similarities of all pairs of nodes. Due to the small-world phenomenon [8], it is unnecessary and is a waste of computation to probe nodes beyond a certain path length (as they are likely the ones already visited anyway). Inspired by the ideas of local-path based methods, we use the hop distance h to restrict the search range, i.e., we expect to obtain the links which are sufficiently far so that recommendations across communities are provided, while calculations related to less relevant links are avoided. This design decision is also in line with the six degrees of separation principle. However, the time complexity of starting from each node to calculate the similarities of all nodes within h hops is still prohibitive. The key issue is how to find the appropriate nodes as starting points. Thus, we propose the concept of *core nodes*. Nodes with high centralities serve as core nodes, where the centralities of nodes are obtained by sampling edges with a biased random walk procedure. The predictive similarity of each pair of nodes is given by a combination of common neighbors and path lengths. We assume that a shorter path identifies a higher similarity to further improve the computational efficiency. We differentiate between the links across communities and those within the same community more explicitly based on the computed similarities. In order to achieve this, we utilize a linear function to transform the computed similarities to final recommendation scores, thereby strengthening the inter-community recommendation scores.

To solve the problem in HeINs, our approach is a novel heterogeneous network embedding method based on random walks and a skip-gram model. We perform random walks following the strategy of JUST [9] and integrate the community properties of nodes into multiple walk sequences. Then, a skip-gram model is trained by treating the sequences of nodes as input. After obtaining the representation vectors of nodes, we calculate the cosine similarities of vectors as the similarities between nodes. Similar to the solution of HoINs, the similarities are transformed to final scores by applying the same linear function.

Our contributions are summarized as follows.

- We demonstrate the significance of link recommendation across communities and formally state the problem. (**Section 3**)

- We devise a novel algorithm based on biased random walks and limited-hop traversal, to efficiently solve the problem of link recommendation across communities. (**Section 4.1**)

- We propose an effective network embedding method based on JUST, which learns the low dimensional representation of nodes, while preserving the community properties in the embeddings. (**Section 4.2**)

- For HoINs, we conduct comprehensive experiments on real-world datasets in various domains, as well as synthetic datasets, and compare our algorithm with neighbor-based, path-based, network-embedding based, and a state-of-the-art link prediction methods, respectively. The experimental results demonstrate that our algorithm outperforms current link prediction methods in both efficiency and accuracy. (**Section 5.1**)

- For HeINs, we evaluate the accuracy and practical recommendation results of our proposed method JSBD. Compared with other recommendation techniques, our method not only shows higher accuracy, but also provides more reasonable recommendations. (**Section 5.2**)

2. Related Work

Link prediction has been extensively studied. Lü et al. [10] and V. Martí et al. [4] presented excellent surveys with experimental comparisons. Current link prediction methods compute similarities for a pair of nodes, and the similarities serve as scores for link prediction. Recommendation systems will recommend links with higher scores. Current link prediction methods compute the similarities in different ways.

Topology information serves as a basis for neighbor-based and path-based methods. For neighbor-based methods, the intuition is that if two nodes have many common neighbors, the likelihood of emerging a link between them is high. The algorithm of common neighbors is the most straightforward way that supports link prediction. A series of extensions from it include Jaccard [11], AA [12], RA [13], and PA [14]. These neighbor-based methods are commonly used to deal with link prediction problems on large scale networks or graph streams. They show acceptable accuracy while maintaining computational efficiency since only two-hop neighbors are considered when calculating similarities. Due to their excellent interpretability and scalability, they have gained wide practical uses. However, the limitation of classic neighbor-based methods is that they are not able to compute the similarities of nodes of a far distance. Particularly, it is hard to explore potential links between nodes across communities.

In contrast to neighbor-based methods, path-based methods reach out to more remote nodes. The line of path-based methods formulate the definition of similarities according to the path information between nodes, and can be categorized into local approaches (such as Katz [15], LPI [16], and LRW [17]) and global approaches (such as SimRank [18], RWR [19], and ACT [17]), where the former usually use a parameter to limit the length of investigated paths so that a low time complexity is achieved, and the latter are unfeasible for large networks owing to their computational complexity. Since path-based methods utilize more information during calculations, they exhibit higher accuracy than neighbor-based techniques at the cost of time. For the problem of link recommendation across communities, we can increase the search range of path-based methods to obtain the similarities of inter-community nodes. However, this results in high time cost and large space consumption.

Probability-based and matrix-factorization based techniques can calculate the similarities between all pairs of nodes. Particularly, probabilistic and statistical methods build models to fit the known structures of the networks, and compute the probability of potential links, which usually works well in some specific scenarios [20, 21, 22, 23]. For example, [22] is suitable for networks of a hierarchical organization, and [23] assumes the degree of clusters of networks is stationary. Therefore, they show ordinary performance when compared with other methods on general datasets. Another shortcoming of such methods is that the training processes are expensive in terms of time cost. Matrix factorization can be applied to learn latent features of nodes, and the predictive similarities can be computed [24] thereafter. Frequently used factorization methods include non-negative matrix factorization and singular value decomposition [25]. They are competitive in terms of accuracy. However, even though networks can be in a compressed sparse matrix form, such methods still suffer from a high computational complexity, and their parallelizations are complex as well. Due to the time complexity of these methods themselves, it is unrealistic to apply them on large datasets.

Recent improvements in machine learning provide new ways for link prediction, treating the problem as a supervised or semi-supervised task. The key of these methods is how to choose appropriate features as the input of classifiers, where topological features and proximity features are commonly used for unlabeled graphs [26, 27, 28]. Original classifiers are directly applicable, such as SVM, KNN, and Decision Tree. There are also a plethora of extensions based on original classification algorithms [29, 30, 31, 32]. These methods may not show a satisfactory performance due to the imbalance of datasets. R. N. Lichtenwalter et al. [27] overcome this problem by the techniques of oversampling and undersampling. These methods inflict a high time-cost because of the training process of classifiers, and their accuracy depends on samples and parameters, which may cause unstable performance.

Moreover, graph embedding and network embedding provide additional feasible ideas for link prediction. They preserve structure information in a low-dimensional space, and each node is represented as a vector [33, 5, 34, 35]. The similarities between nodes can be easily estimated by the inner product or the cosine similarity. Particularly, [5] can learn representations that organize nodes based on their network roles and communities they belong to, providing a clear distinction of links within or across communities, thus could best fit our purpose. Evaluation results also show it has surprisingly good performance, but not as good as our method. Moreover, [35] proposes a state-of-the-art link prediction framework based on graph neural networks. It preserves information related to links by local subgraphs and learns heuristics using a graph neural network. Detailed experimental comparisons show that [35] performs better than some heuristic methods and latent feature methods. In addition, all the above methods lack attention to inter-community links.

Since HeINs will degenerate into HoINs when not considering the types of nodes and edges, all the methods mentioned above can be directly applied to solve the problem of link recommendation in HeINs. However, ignoring the node types of HeINs causes significant information loss. It is necessary to model different nodes and relationships in a unified form. [36] predefines meta-paths to guide heterogeneous random walks and capture different semantic information. Huang et al. [37] utilize dynamic programming to calculate the similarities of truncated meta-paths and adopt a similar strategy as [38] to preserve the similarities in a low-dimensional space. Fu et al. [39] propose a heterogeneous network embedding method Hin2vec, which learns the potential vectors of nodes and meta-paths by specifying the relationship set and performing several link prediction tasks together. A main drawback of methods based on meta-paths is that the choice of meta-paths has a large impact on the embedding results, which requires more prior knowledge in the application field to choose the appropriate meta-paths. Hussenin et al. [9] balance the proportion of homogeneous edges and heterogeneous edges using the jump and stay strategies, thereby overcoming the weakness of meta-path techniques. Tang et al. [40] propose a heterogeneous representation learning method PTE based on LINE [38]. It divides a heterogeneous network into several bipartite graphs, then performs the embedding method of LINE on each graph. The final embeddings of nodes are obtained by jointly optimizing the linear combination of objective functions. Xu et al. [41] design a PME method to capture the first-order and second-order proximities in a unified form based on metric learning.

In addition to the aforementioned studies, some remotely related studies exist in the literature. For example, multiple methods related to community recommendation are proposed [42, 43, 44]. However, there are fundamental differences between them and our work. We provide entity-to-entity recommendations across communities; thus, our recommendations are more specific. [45] considers the evolution of structure within online social networks and presents a series of measurements, thereby dividing the networks into three regions. However, this kind of segmentation is too broad for our study.

3. Problem Statement

Consider Examples 1 and 2 in Section 1. We formulate the problem of link recommendation across communities as follows.

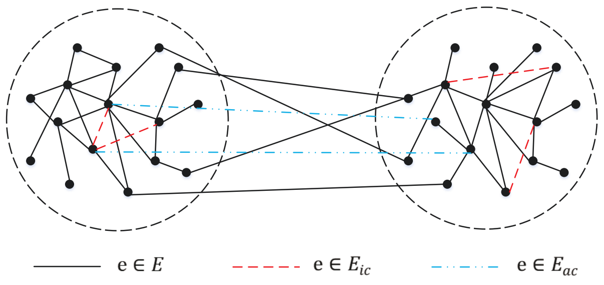


Figure 2: Illustration of the problem of link recommendation across communities in HoINs. The dashed ovals represent two communities.

Problem 3.1. ***(Link Recommendation Across Communities in HoINs)*** *Given an undirected network as input represented as $G = (V, E)$, where V is the set of nodes, and E is the set of links. A community C refers to a subset of vertices, i.e., $C \subset V$. For two communities C_1 and C_2, it holds that $|E_{C_1}| \gg |E_{C_1,C_2}|$ and $|E_{C_2}| \gg |E_{C_1,C_2}|$, where $|E_{C_1}|$ (resp. $|E_{C_2}|$) is the number of edges between two nodes in C_1 (resp. C_2), and $|E_{C_1,C_2}|$ is the number of edges between a node in C_1 and a node in C_2. We use Figure 2 to illustrate the problem that we study. The dashed ovals enclose two communities. The edges in the same community are much denser than those across communities. Let the set of potential links across communities and that within the same community be E_{ac} and E_{ic}, respectively. Link recommendation tells the likelihood/scores of potential edges in the set $V \times V - E$. Existing link prediction approaches typically give higher scores for the edges in E_{ic}. To extend the range of recommendations,* ***Link recommendation across communities in HoINs*** *focus on how to methodologically shift emphasis on E_{ac} rather than E_{ic}.*

Definition 3.1. ***(Heterogeneous Information Networks) A heterogeneous information network*** *can be represented as $G = (V, E, \mathcal{R}, \mathcal{T})$ in which V is the set of nodes, and E is the set of relationships. $\mathcal{R}$ and $\mathcal{T}$ are* *the type sets of V and E, respectively, where $|\mathcal{R}| + |\mathcal{T}| > 2$. For $v \in V$, it is mapped to $\mathcal{R}$ by a function $\phi(\cdot)$, i.e., $\phi(v) : V \to \mathcal{R}$. E and $\mathcal{T}$ are associated with a mapping function $\varphi(\cdot)$, i.e., for $e \in E$, $\varphi(e) : E \to \mathcal{T}$.*

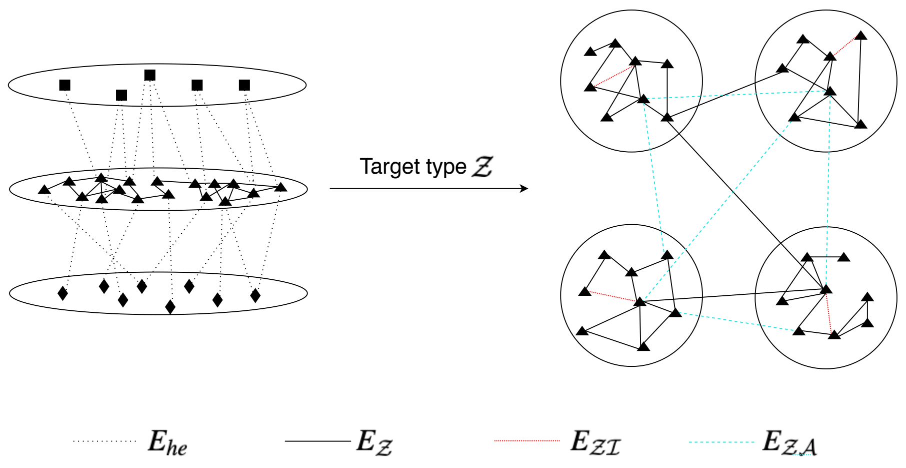


Figure 3: Illustration of the problem of link recommendation across communties in HeINs.

Problem 3.2. ***(Link Recommendation Across Communities in HeINs)*** *Given a heterogeneous information network $G = (V, E, \mathcal{R}, \mathcal{T})$, let $\mathcal{Z}$ be the target entity type for recommendation. $V_{\mathcal{Z}}$ is the set of $\mathcal{Z}$-type nodes. As shown in Figure 3, $E_{\mathcal{Z}}$ is the set of edges incident to two Z-type nodes. $E_{\mathcal{ZA}}$ represents the set of potential inter-community links which are generated by nodes in $V_{\mathcal{Z}}$, while $E_{\mathcal{ZI}}$ is the set of potential intra-community links. Similar to Problem 3.1,* ***link recommendation across communities in HeINs*** *studies how to methodologically shift emphasis on $E_{\mathcal{ZA}}$ rather than $E_{\mathcal{ZI}}$.*

4. Main Algorithm

We now propose the solutions of link recommendation across communities in HoINs and HeINs. In the following, we present our approaches for the two network models, respectively.

4.1. The Proposed Approach on HoINs

Recall Example 1 in Section 1. Consider a collaboration network where different research fields represent different communities. There are some active researchers who are well known in the field s/he belongs to and who are therefore also known by the outsiders. Hence, the chance of potential links across fields around them is higher. We aim to find these people, denoted as *core nodes* in the network, to perform the exploration of links across communities starting from them.

An intuition is to measure the centrality and the role of a node by calculating an index. However, current centrality measurements fall short, in that the degree centrality [46] cannot capture the community structures, while the betweenness centrality [47] incurs a high time cost. The notion of core-periphery structure has been studied for long. For a core-periphery network, cores are surrounded by peripheral nodes, which are not connected to each other [48]. There are two main models behind the definition of core-periphery networks; one assumes that a network contains only one core, while the other allows the existence of multiple cores [49]. Particularly, [50] assigns a coreness value to each node by random walkers, thus profiling core-periphery network structures. However, for this kind of network model, the definition of a core node is different from ours. The former only emphasizes the connectivity of a node within a module, whereas the latter additionally takes the "bridge" function of a node into consideration. It is not reasonable to apply the techniques of core-periphery networks directly. Therefore, we simulate a biased random walk procedure to automatically and efficiently obtain the core nodes.

In addition, to recommend potential collaborators (in Example 1), we do not need to calculate the similarities between one researcher and all other researchers in the network since the fields of some researchers exhibit significant differences, and it is hard to generate a collaboration. Thus, we utilize a *limited-hop traversal* to narrow down the calculation range and thereby reduce the time consumption. Given the competitive performance of neighbor-based methods, the similarity of a pair of nodes is computed from the combination of the common-neighbor method and the path length.

Since connections within the same community are denser, the similarity scores of these links are often higher than those across communities. Therefore, we use a linear function to strengthen the inter-community link recommendation scores. The resulting approach achieves competitive accuracy and remarkable efficiency, in contrast to prior work.

The whole algorithm includes the following four steps:

- Use biased random walks to sample edges, adjusting the parameter to balance the proportion of BFS and DFS.

- Compute the centralities of edges and further obtain the centralities of nodes. Select core nodes based on the centralities of all nodes.
- Start from each core node, and perform a limited-hop traversal to compute the similarities of each pair of nodes in the range.
- Prioritize the inter-community link recommendation scores by a linear function.

4.1.1. Biased Random Walk

We use a random walk procedure for sampling edges. BFS [51] and DFS [52] are two biased strategies for the walk. As two common traversal methods of a graph, they are capable of capturing different features of nodes. BFS tends to capture the degree feature of nodes, while DFS reflects how important a node is in contributing to the connectivity of a graph. Only considering one of them will miss certain features of nodes. Therefore, we propose a biased random walk procedure to incorporate the two sampling methods together. The detailed random walk procedure shown in Algorithm 1 is described as follows.

Given a random walk procedure r, suppose the k-th intermediate node is r_k, we use $\Gamma(v)$ to denote the neighbors of v. During each procedure, we ensure that each node is *visited* at most once, where the step of returning to r_{i-1} (from r_i) to probabilistically reach other neighbors of r_{i-1} is not counted as a revisit of r_{i-1}. Given the set of visited nodes VN and the current node r_i, the next step r_{i+1} is determined by the following transition function:

$$P(r_{i+1} = r_{i-1} | r_i) = 1 - \alpha, \tag{1}$$

$$P(r_{i+1} \in \Gamma(r_i) \setminus VN) = \alpha \tag{2}$$

assuming $\Gamma(r_i) \setminus VN \neq \varnothing$, where α is a parameter to control the bias between BFS and DFS, guiding the random walk. A larger α represents a higher DFS propensity. If $\Gamma(r_i) \setminus VN = \varnothing$, then $P(r_{i+1} = r_{i-1} | r_i) = 1$. For the starting node $r_0 = u$, we have $P(r_1 \in \Gamma(u) \setminus VN | r_0 = u) = 1$.

As we aim to solve the problem of mining potential links across communities, we set α to be greater than 0.5 to enhance the effect of DFS. Algorithm 1 uses l to limit the number of steps of a random walk procedure. The function *RandomNeighbors* used in lines 3 and 8 chooses an unvisited neighbor

Algorithm 1: RandomWalk (G, u, l, α)

```
   Input: $G = (V, E)$: homogeneous information network,
   $u$: starting node of a random walk,
   $l$: upper bound of number of steps
   $\alpha$: bias parameter
1  $i \leftarrow 0$, $r_i \leftarrow u$, $visited \leftarrow \{u\}$
2  while $i < l$ do
3      if RandomNeighbors($r_i$, $visited$) = NULL then
4          $r_i \leftarrow r_{i-1}$
5          continue
6      $p \leftarrow$ Random(0, 1)        //obtain a random number in (0, 1)
7      if $p$ ¡ $\alpha$ or $i = 0$ then
8          $r_i \leftarrow$ RandomNeighbors($r_i$, $visited$)
9          $i \leftarrow i + 1$
10     else
11         $r_i \leftarrow r_{i-1}$
12     $visited \leftarrow visited \cup r_i$                 //mark $r_i$ as visited
```

uniformly at random. Lines 7-9 are leading towards DFS while lines 10-11 towards BFS.

Figure 4 is a simple simulation to illustrate the next one or two steps of a random walk procedure r. The walker currently resides at r_i. So it has probability $1 - \alpha$ to return to r_{i-1} and probability α to move to its other neighbors. The chance that it reaches one of those neighbors depends on its degree and the number of already-visited neighbors. Suppose it chooses r_{i-1} as the next step. Then the subsequent probability to again go back to u is also $1 - \alpha$. Therefore, the probability that the walk reaches u in two steps from now is $(1 - \alpha)^2$.

4.1.2. Core Nodes Filtering

We say that an edge is sampled when a random walk traverses through it (an edge is sampled at most once in each random walk procedure). Through repeating a large number of random walk procedures with different starting nodes, we accomplish an edge sampling process. Let *iter* denote the number of iterations of the random walk procedures, and the starting node is randomly chosen in each iteration. The weights of all edges are initialized to 1.

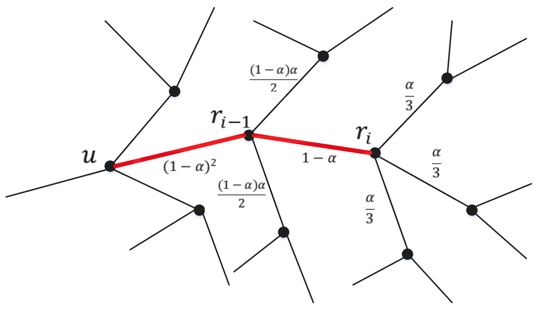


Figure 4: Illustration of the biased random walk, which is currently at r_i. The numbers at each edge are the probabilities of those edges being visited in one or two steps. The two edges in red are the ones visited in the past.

The weight of an edge is increased by 1 every time the edge is sampled. The final weight is considered as the edge's centrality. In this way, we are capable of assigning higher weights for edges that play key roles in supporting the connectivity of the network. To overcome the problem that the gap between the maximum centrality and the minimum centrality may be large, we normalize the centrality of each edge by a log function. The final centrality of an edge e is defined as

$$Cen_e(e) = \frac{\lg(w_e)}{\lg(w_{max})},$$

where w_e is the weight of edge e, and w_{max} is the maximum weight of all edges. $Cen_e(e)$ lies in [0, 1].

We then use the centralities of edges incident at a node to compute the centrality of that node. Intuitively, the nodes connected to edges with greater weights are more important, and the degree reflects the roles of nodes as well. Hence, we define node centrality as the sum of centralities of edges connected to the node. Likewise, the node centralities are normalized by the same log function. The centrality of a node v is computed as

$$Cen_v(v) = \frac{\lg(\sum_{e\in\{(u,v)|u\in\Gamma(v)\}} Cen_e(e))}{\max\limits_{v\in V} \lg(\sum_{e\in\{(u,v)|u\in\Gamma(v)\}} Cen_e(e))},$$

where the denominator takes the maximum over all nodes. $Cen_v(v)$ is also in [0, 1].

We use $Cen_v(v)$ as a metric for the importance of nodes. Given a threshold interval $[\epsilon_1, \epsilon_2]$, the core node set consists of nodes whose centralities lie in the interval. As a result, for an undirected graph, the size of its core node set can be controlled by tuning ϵ_1 and ϵ_2. Extremely high centralities are typically associated with the highest degree nodes in dense regions of core nodes with overlapping h-hop traversal ranges (Sec. 4.3), making them redundant as core nodes. Hence, we choose an interval rather than a single threshold to filter the core nodes, aiming to increase the diversity of core nodes and reduce the computational cost of the subsequent limited-hop traversal.

4.1.3. Limited-hop Traversal

Considering the high effectiveness of neighbor-based methods and the short distance limitation, we propose a novel method of computing the similarity of a pair of nodes, which is based on the Salton index [53] as well as the paths between two nodes. There are many neighbor-based techniques of link prediction, such as Jaccard [11], AA [12], RA [13], and PA [14]. We choose Salton index as a foundation to define the final similarity of a pair of nodes because it shows the best average performance for general link prediction tasks that we perform on multiple complex networks. For a pair of nodes (x, y), the Salton index is defined as

$$Salton_{xy} = \frac{|\Gamma(x) \cap \Gamma(y)|}{\sqrt{|\Gamma(x)||\Gamma(y)|}}.$$

Another intuition is that the similarity decreases as the distance between the two nodes increases. Therefore, for a path from x to y, we calculate the product of the Salton index of pairs of nodes in the order in the path, and the maximum value among all paths is the final similarity between nodes (x, y). This can be formulated as

$$s_{xy} = \max_{path:x\to y} \prod_{i=0}^{i=|path|-1} Salton_{v_i v_{i+1}} | v_i \in path,$$

where $path$ is a sequence of edges from x to y, and v_i is the i-th node on the path. In particular, we have $v_0 = x$ and $v_{|path|} = y$. The max is taken over all paths from x to y.

Next, starting from each core node, we explore nodes within h hops by BFS and assign an increasing level for them. In a bottom-up fashion, we obtain similarities for all pairs of each i-level node and its $(h-i)$-hop reachable nodes. We visit nodes in decreasing order of levels. The similarity calculation process of each node within the exploration range has two phases. For a node v, the first phase collects the computed similarities from v's higher level neighbors, and the second phase computes the similarities between v and nodes which are reachable through those located in the same level as v.

There is no need to compute the similarities between v and v's lower level neighbors. Suppose v is an ith-level node, and u is a jth-level node that is $(h-i)$-hop reachable from v, where $j < i$. Since the exploring range of u is $(h-j)$-hop, we will access v and obtain the similarity of $(u$,$v)$ when calculating the similarities of u's related pairs of nodes. The limited-hop process terminates when we reach the current core node.

It is possible that other core nodes appear in the exploration range of the current core node. For a core node v located at the ith level of another core node u, during the exploration of u, the nodes within $(h-i)$-hops of v will be explored as well. Hence, we choose the lowest-level core node as the starting node for the next limited-hop traversal. If there are no other core nodes in the current exploration range, we randomly pick the next core node.

In addition, we use two pruning strategies to reduce the computation cost. We use Figure 5 as a simple example to illustrate them. One of the pruning strategies is based on an assumption that a shorter path between two nodes generally denotes a higher similarity. As shown in Figure 5 (a), the similarity of $[v_1,\ v_3]$ calculated by the path $< v_1, v_2, v_3 >$ is assumed to be higher than that calculated by the path $< v_1, v_5, v_4, v_3 >$. The second strategy is that if one node has been accessed and the similarity is higher than the current path, the subsequent calculation starting from that node are pruned. Figure 5(b) illustrates this situation. Suppose v_4 has been visited by the path $< v_1, v_2, v_3, v_4 >$. When it is later visited by the path $< v_1, v_6, v_5, v_4 >$, if the new similarity of $[v_1,\ v_4]$ is smaller than the current one, then subsequent visits will always skip the path $< v_1, v_6, v_5, v_4 >$.

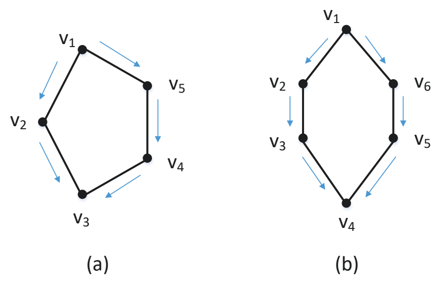


Figure 5: Illustration of two pruning strategies.

Algorithm 2: LIMITED-HOP (G, c, h)

```
Input: $G = (V, E)$: homogeneous information network,
  $c$: starting core node of the limited-hop traversal,
  $h$: upper bound of the limited-hop traversal
Output: $similarities$: the similarities of node pairs within $h$ hops from $c$
1  Start from $c$ and fill $\mathcal{L}_j$ from 0 to $h$ by BFS
2  foreach $u$ in $\mathcal{L}$ do
3      $reachAll[u]$, $directHigh[u]$ ← empty list
4  foreach $u$ in $\mathcal{L}_h$ do
5      $reachAll[u]$.add($u$)
6  for $j \leftarrow h$ - $1$ to 0 do
7      Initialize $directReach[m]$ to empty list for $m \in \mathcal{L}_j$
8      foreach $u$ in $\mathcal{L}_j$ do
9          foreach $v$ in $\Gamma(u)$ do
10             if $l[v]>l[u]$ then
11                 Add $v$ to $reachAll$, $directHigh$ and $directReach$
12                 foreach $x$ in $reachAll[v]$ do
13                     updateSimilarity($x$, $u$, Salton$_{uv}$ · $similarities[x, v]$)
14                     $reachAll[u]$.add($x$)
15                     if $x$ in $directReach[v]$ then
16                         $directReach[u]$.add($x$)
17     foreach $u$ in $\mathcal{L}_j$ do
18         $nbr1$, $nbr2$, $sameLevel$ ← empty list
19         $nbr1$.add($u$)
20         for $dis \leftarrow 1$ to $h - j$ do
21             foreach $x$ in $nbr1$ do
22                 foreach $y$ in $\Gamma(x)$ do
23                     $nbr2$.add($y$)
24                     if $l[y]$==$l[u]$ && $y$ not in $sameLevel$ then
25                         Add $y$ to $currentHop$, $sameLevel$ and $reachAll$
26                     updateSimilarity($u$, $y$, $similarities[u, x]$· Salton$_{xy}$)
27             $nbr1$ ← $nbr2$, $nbr2$ ← empty list
28             foreach $x$ in $currentHop$ do
29                 CALSIMTHROUGHSAMELEVEL($u$, $x$, $h - j - dis$)
```

The details of the proposed method are described in the Algorithm LIMITED-HOP. A core node c serves as the starting node of the limited-hop process, and the exploration range is limited by h. Particularly, for a level j, $\mathcal{L}_j$ holds the set of nodes located at the jth level. And for a node v, $l[v]$ represents the level of v. Line 1 fills the $\mathcal{L}$ lists within h hops using BFS starting from c. Then, we maintain two lists for each node in $\mathcal{L}$ to record their $(h-j)$-hop reachable nodes and their higher-level neighbors, which are denoted as *reachAll* and *directHigh*, respectively. These lists are created and initialized to be empty in lines 2 and 3. Then, lines 4 and 5 add the highest level nodes to the *reachAll* lists of themselves. The loop from line 6 iteratively computes the similarities between the nodes and their limited-hop reachable nodes from the $(h-1)$-level nodes to c in descending level order. During the iteration of each level, in order to prune unnecessary calculations, we record the nodes whose shortest paths from the current computing node are known in *directReach*, utilizing our pruning strategy 1. The corresponding lists are created and initialized in line 7.

The remaining algorithm is divided into two phases. Let u be current node, and the level of u is j. In lines 8-16, for each neighbor node v in $\Gamma[u]$ with a level greater than u, we collect computed similarities between v's $(h-j-1)$-hop reachable nodes (stored in *reachAll*$[v]$) and v itself, and then multiply each similarity with the similarity of the pair of nodes $[u,v]$ (i.e. the Salton value) to get the similarities between u and u's reachable nodes through its higher-level neighbors. In line 11 and lines 15-16, *reachAll*$[u]$, *directHigh*$[u]$, and *directReach*$[u]$ are updated along with the calculation. The function *updateSimilarity* called in lines 13 and 26 is used to update the similarity of its first two parameters, i.e., if the current value is smaller than the third parameter, update the similarity with the third parameter. In lines 17-29, the second phase calculates the similarities between nodes which are reachable through nodes in $\mathcal{L}_j$ and u. Starting from u, we explore nodes in $\mathcal{L}_j$ hop by hop, where the upper bound is $(h-j)$ hops. The lists *nbr*1 and *nbr*2 maintain the $(dis-1)$-hop and *dis*-hop reachable nodes, respectively. We use *sameLevel* to record the same-level nodes that have been visited and *currentHop* stores the newly explored same-level nodes in line 25. For a node that takes *dis* hops to arrive, the remaining hop count is $h-j-dis$. We use the algorithm CALSIMSTHROUGHSAMELEVEL to obtain the similarities in the subsequent visits starting from those same-level nodes. The process of exploring nodes in $\mathcal{L}_j$ will not cause a high time cost. We terminate it when all nodes in $\mathcal{L}_j$ (except u itself) have already been added into *sameLevel*. The

Algorithm 3: CalSimThroughSameLevel (u, x, hop)

```
Input: $u$: current node,
$x$: a node in the same level as $u$,
$hop$: remaining hops from x
1  Initialize $steps[]$ to $\infty$ for each node
2  q $\leftarrow$ a priority queue
3  foreach $z$ in $directHigh[x]$ do
4      $q$.add($z$)
5      $steps[z] \leftarrow 1$
6  while ! $q$.isEmpty() do
7      $y \leftarrow q$.poll()
8      if $steps[y] \geq hop$ then
9          break
10     foreach $z$ in $\Gamma[y]$ do
11         if $steps[z] > steps[y]$ then
12             if $z$ not in $directReach[u]$ then
13                 updateSimilarity($u, z, similarities[u, y] \cdot similarities[y, z]$)
14                 if $z$ not in $q$ then
15                     $q$.add($z$)
16                     $reachAll[u]$.add($z$)
17                     $steps[z] = steps[y] + 1$
```

search range is small for higher level nodes, and c provides a short reachable path for lower level nodes. Algorithm 2 keeps running until the iteration reaches c. Finally, all the calculated similarities are stored in *similarities*.

Algorithm 3 presents how the similarities between nodes and those reachable through nodes located at the same level are computed. To control the range of remaining hops, we maintain a list *steps* and initialize it to infinity (maximum value) for each node in line 1. During the subsequent calculation, we explore nodes according to *steps* and record the access order, which can be done by using a priority queue q. In the first step of exploration, we only visit x's higher level neighbors, which is shown in lines 3-5. Next, for each node in q, as shown in lines 10-17, we visit all of its neighbors z, and update the similarity of (u, z) when z is not accessed by a shorter path, utilizing our pruning strategies 1 and 2. If q does not contain z, we add z into q and

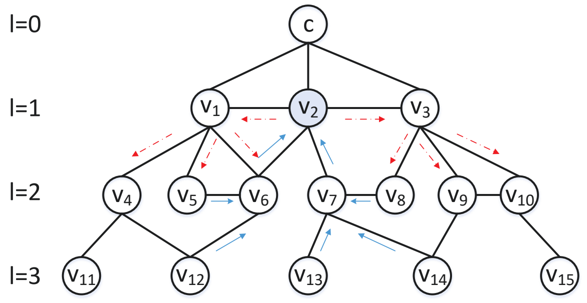


Figure 6: Illustrating the limited-hop process. The levels of nodes are shown on the left. c is the starting core node of the limited-hop process. Solid arrows represent the similarities that can be calculated from higher-level neighbors of v_2. Dotted arrows denote the propagation of calculations starting from the same-level nodes within $h-1$ hops.

modify $reachAll[u]$ and $steps[z]$.

We use Figure 6 to illustrate how Limited-Hop and CalSimThrough-SameLevel work. Suppose c is a core node and the nodes below it are reachable from c within $h = 3$ hops, located at the corresponding levels of BFS. We start the entire calculation from the $level = 2$ nodes. Suppose we have finished the calculation of nodes in $\mathcal{L}_2$ and are calculating the similarities related to v_2. Firstly, we visit its higher-level neighbors v_6 and v_7. v_6 and v_7 should have already performed 1-hop accesses, i.e., the similarities of $[v_6, v_{12}]$, $[v_6, v_5]$, $[v_7, v_{13}]$, $[v_7, v_{14}]$, and $[v_7, v_8]$ are known. Hence, we just multiply them with similarities$[v_2,\ v_6]$ and similarities$[v_2,\ v_7]$, respectively. The $directReach[u]$ is $\{v_6,\ v_7,\ v_{12},\ v_{13},\ v_{14}\}$. Next, we handle the same-level neighbors v_1 and v_3. Due to the two-phase algorithm, the similarities related to their high-level neighbors have been obtained before. similarities$[v_2,\ v_1]$ and similarities$[v_2,\ v_3]$ are calculated during the process of exploring the same-level neighbors of v_2. The remaining hop of v_1 and v_3 is equal to 1. $\{v_4,\ v_5,\ v_6\}$ is the explored set starting from v_1, while v_6 is in $directReach[u]$, so v_6 and $reachAll[v_6]$ will not be accessed again. In the situation of Figure 6, v_6 has already been the upper bound of limited hops passing through v_1. For v_3, the related nodes are $\{v_8,\ v_9,\ v_{10}\}$. Hence, we multiply similarities$[v_2,\ v_3]$ with similarities of $[v_3,\ v_8]$, $[v_3,\ v_9]$, and $[v_3,\ v_{10}]$. The similarities of all 2-hop reachable pairs of nodes starting from v_2 have been calculated now. Similar processes are performed on v_1 and v_3. The core node c just needs to collect similarities from its higher-level neighbors, and then, we finish the limited-hop process starting from c with $h = 3$.

4.1.4. Inter-Community Links Strengthening

The connections within the same community are denser than those across communities. As a result, it is natural that the similarities of links within the same community are often higher. Therefore, after obtaining the similarities of pairs of nodes, we assume there is a threshold δ, such that pairs of nodes with similarity greater than δ tend to be in the same community, while the other pairs tend to be across communities. While a perfect choice of δ may not exist, we resort to an approximate partition. We propose a linear function to boost the inter-community scores. Combined with Area Under Curve, it is a measure of evaluating how well a method distinguishes between links across communities and those within a community. The final recommendation score is defined as:

$$Score_{xy} = \begin{cases} Similarity_{xy}, & \text{if } Similarity_{xy} \leq \delta \\ 2\delta - Similarity_{xy}, & \text{if } Similarity_{xy} > \delta \end{cases}$$

where δ is estimated from the average similarity of all pairs of nodes, making it adaptive to different datasets.

4.1.5. Complete Algorithm

Algorithm 4 shows the complete algorithm. In lines 1-3, we sample edges through repeating the biased random walk procedure *iter* times, where the function *RandomNode* selects a node randomly as the starting node of the current random walk. According to the sampling results, line 4 calculates the node centralities using the formulas proposed in Section 4.1.2 and composes the core node set $\mathcal{C}$. Next, multiple threads start to compute the link recommendation scores. In our experiments, the number of threads is set to 4.

The variable *cNext* is used to store the next core node to be explored. It is initialized to -1 in line 6, implying that the next core node is unknown. The loop from line 7 calculates the similarities following the exploration order of the core nodes. When the next core node is unknown (i.e., the condition holds in line 10), line 11 selects a core node in $\mathcal{C}$ randomly. In lines 13-14, the limited-hop process starts from c, and then the next core node is determined, where the function *NextCore* assigns the core node with the lowest level in the current exploration range to *cNext*. Once *cNext* stores a value other than -1, lines 16-18 are performed. Since the similarities related to *cNext* are available, we modify the loop in line 8 of LIMITED-HOP slightly and denote it as LIMITED-HOP*. We terminate the loop when the iteration of $j = 1$

```
Algorithm 4: CNLH $(G, l, iter, \alpha, \epsilon_1, \epsilon_2, h, \delta)$
  Input: $G = (V, E)$: homogeneous information network,
  $l$: walk length of the random walk,
  $iter$: number of random walk iterations,
  $\alpha$: bias parameter of the random walk,
  $\epsilon 1$: lower bound of determining core nodes,
  $\epsilon 2$: upper bound of determining core nodes,
  $h$: upper bound of the limited-hop traversal,
  $\delta$: threshold to enhance inter-community links
  Output: $scores$: scores of all calculated node pairs
1  for $i \leftarrow 0$ to $iter$ do
2      $u \leftarrow$ RandomNode()
3      RandomWalk$(G, u, l, \alpha)$
4  $\mathcal{C} \leftarrow$ FilterCore($\epsilon_1$, $\epsilon_2$)
5  foreach $thread$ do
6      $cNext \leftarrow$ -1
7      while $true$ do
8          if $\mathcal{C}$.isEmpty() then
9              break;
10         if $cNext$ == -1 then
11             $c$ = RandomCore()
12             $\mathcal{C}$.remove($c$)
13             Limited-Hop$(G, c, h)$
14             $cNext$ = NextCore()
15         else
16             Limited-Hop*$(G$, $cNext$, $h)$
17             supplement the new explored similarities for cNext
18             $cNext$ = NextCore()
19 $scores$ = EnhanceICLinks($similarities$, $\delta$)
20 return $scores$
```

finishes, i.e., we do not collect those explored similarities of $cNext$. Suppose the level of $cNext$ is i in the last limited-hop exploration, the similarities between $cNext$'s $(h - i)$-hop reachable nodes and $cNext$ itself have been computed; so we just need to supplement the remaining i-hop similarities, which are stored in the *reachAll* structure of $cNext$'s $(h-i)$-level nodes. We do not reuse the calculated similarities of other nodes explored in the last limited-hop because the relative levels of the neighbors of a node may have changed in the new limited-hop traversal and only the remaining uncalculated similarities of $cNext$ are directly available. *EnhanceICLinks* called in line 19 transforms each similarity into final recommendation scores using the linear function described in Section 4.1.4. The loop from line 7 keeps running until the condition in line 8 is met, which indicates all core nodes have been explored. Our algorithm stops when all threads finish running.

4.2. The Proposed Approach on HeINs

In this section, we propose a heterogeneous network embedding method based on the random walk strategy JUST and the skip-gram model. We integrate community features into the processes of random walks, and then generate the final node sequences and learn the representation vectors of nodes by training a skip-gram model.

4.2.1. Node Sequence Generation

In order to generate the node sequences, we perform a random walk process as follows. We first adopt the walk strategy in JUST [9] to determine the node type of the next hop. There are two options to select the type of next node: stay in the same type with the current node or jump to one of different types. Suppose the type of the current node v_i is $\mathcal{I}$. Let $\mathcal{R}$ be the set of all node types, and $V_{\mathcal{I}}(v_i)$ be the set of neighbors of the same type as v_i. Given a type $\mathcal{P}$, we use $V_{\mathcal{P}}(v_i)$ to denote the $\mathcal{P}$-type neighbors of v_i. For v_{i+1}, the probability of staying in $\mathcal{I}$ can be defined as:

$$P(\phi(v_{i+1}) = \mathcal{I}) = \begin{cases} 0, & \text{if } V_{\mathcal{I}}(v_i) = \varnothing \\ 1, & \text{if } \{V_{\mathcal{P}}(v_i) | \mathcal{P} \in \mathcal{R}, \mathcal{P} \neq \mathcal{I}\} = \varnothing \\ \alpha^l, & \text{otherwise} \end{cases} \tag{3}$$

where α is the parameter of controlling the inital stay probability, and l is the length of the consecutive node sequence ending at v_i and having the same type as v_i.

If the current random walk process chooses to jump to a new type, the specific type of the next node needs to be selected in advance. The process randomly samples a type from those differing from the last m visited types. Let $\mathcal{R}_{lm}$ denote the last m visited types (including the current node type $\mathcal{I}$). Let us consider a candidate set of node types where each type is not in $\mathcal{R}_{lm}$ and there are v_i's neighbors that belong to the type. If this set is not empty, the type of the next node will be sampled from it uniformly at random; otherwise the process will ignore the limitation of $\mathcal{R}_{lm}$. Therefore, the candidate set of available types for the next hop can be expressed as:

$$R_{he}(v_i) = \begin{cases} \{\mathcal{P} | \mathcal{P} \in \mathcal{R} \wedge \mathcal{P} \notin \mathcal{R}_{lm}, V_{\mathcal{P}}(v_i) \neq \varnothing\}, & \text{if not empty} \\ \{\mathcal{P} | \mathcal{P} \in \mathcal{R}, \mathcal{P} \neq \mathcal{I}, V_{\mathcal{P}}(v_i) \neq \varnothing\}, & \text{otherwise} \end{cases} \tag{4}$$

After determining a specific type $\mathcal{Q}$, we selected nodes with a bias to capture community features during the random walk, i.e., for the next hop, we may preferentially sample the common neighbors of the current node and the previously visited nodes. We will extend the search range of the common neighbors since there might be no edges between some node types in HeINs. Considering that the probability of community associated with the current node decreases as the exploration range increases, we introduce an exponential function to weaken the bias probability. Suppose a visited node v_d is d-hop away from the current node v_i. V_{comm} refers to the set of the common neighbors of v_d and v_i, and the type of all nodes in V_{comm} is $\mathcal{Q}$. If V_{comm} is not empty, the probability that the next hop randomly selects a node in V_{comm} is:

$$P(v_{i+1} \in V_{comm}) = \beta^{d-1} \tag{5}$$

where β is the bias parameter. We start the exploration from v_i along with the reverse order of the generated node sequence. A parameter k is used to limit the exploration range. The process is terminated when we find a visited node making V_{comm} non-empty. If there is no node that meets the condition, the random walk will sample a node in $V_{\mathcal{Q}}(v_i)$ randomly.

The detailed walk strategy is described in Algorithm 5. The loop from line 2 iteratively samples a new node until the length of the current walk sequence is l_{max}. Let *nextType* hold the type of the next node, and it is initialized to the type of the current node. During the iteration, we update *nextType* according to Equations 3 and 4. We use y as the index of the explored node, and the loop from line 9 starts the exploration from $y = k$

Algorithm 5: JSBDRandomWalk($G, v, l_{max}, \alpha, m, \beta, k$)

```
Input: $G = (V, E, \mathcal{R}, \mathcal{T})$: heterogeneous information network,
$v$: staring node of the random walk,
$l_{max}$: maximum length of the random walk,
$\alpha$: initial stay parameter,
$m$: number of memoried types,
$\beta$: initial bias parameter,
$k$: upper bound of exploration range
Output: $rw$: a sequence of nodes
1  $rw \leftarrow \{v\}$, $R_{lm} \leftarrow \{\phi(v)\}$
2  while $|rw| < l_{max}$ do
3      $nextType \leftarrow \phi(v)$
4      Decide to walk along with $E_{he}$ or $E_{ho}$ according to Equation 3
5      if with $E_{he}$ then
6          Sample a type $\mathcal{P}$ from $R_{he}(v)$ according to Equation 4
7          $nextType = \mathcal{P}$
8      Update $R_{lm}$ with $nextType$
9      for $y \leftarrow k$ to $1$ do
10         if $y < |rw|$ and $\{u|u \in \Gamma(rw[y]) \cap \Gamma(v), \phi(u) = nextType\} \neq \varnothing$ then
11             $candidates = \{u|u \in \Gamma(rw[y]) \cap \Gamma(v), \phi(u) = nextType\}$
12             $biasedNode$ = RandomPick($candidates$)
13             break
14     calculate the probability $p$ according to Equation 5
15     if $Random(0, 1) > p$ and $biasedNode \neq$ NULL then
16         $nextNode = biasedNode$
17     else
18         $nextNode$ = RandomPickNeighbors($v$, $nextType$)
19     if $nextNode$ == NULL then
20         break
21     $rw$.add($nextNode$);
22 return $rw$
```

to 1. The common neighbors will be recorded in *candidates* once we find a qualified node (i.e., the condition holds in line 10). The function *RandomPick* randomly selects a node in *candidates* as the candidate biased node. Next, we calculate the bias probability according to Equation 5. Lines 15-16 imply that the random walk selects *biasedNode* as the next hop, while lines 17-18 indicate that we will sample a *nextType*-domain neighbor of v by the function *RandomPickNeighbors*. If there is no suitable $nextNode$, the current random walk will be terminated. Finally, the sequence of nodes generated by a complete random walk is maintained in rw.

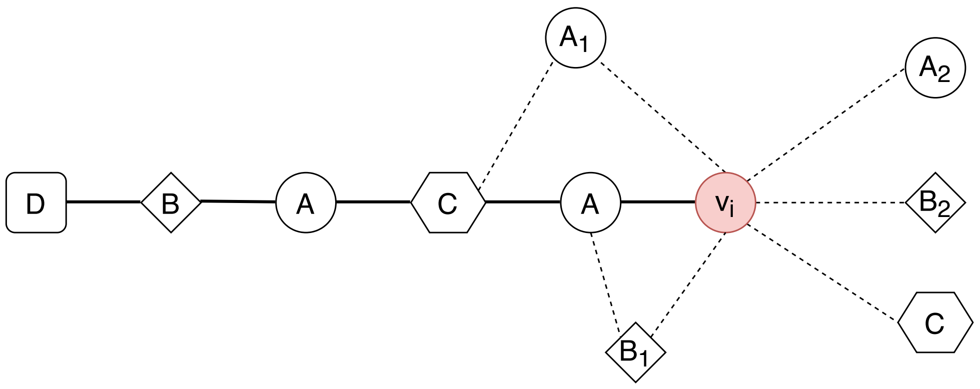


Figure 7: The generation process of node sequences

We use Figure 7 as an example to illustrate how the random walk strategy is performed. Suppose the type of v_i is A, and $m = 2$. The solid lines represent the generated node sequence, and the dotted lines are other edges in the heterogeneous graph. The current random walk rw has probability α^2 to jump to a A-type node and probability $\frac{1-\alpha^2}{2}$ to move to a B-type or C-type node. Due to $R_{lm} = \{A, C\}$, it is impossible that rw selects a C-type node as the next hop. If rw chooses to stay in the same type as v_i, it has probability β to move to A_1 and probability $1 - \beta$ to move to A_2. The next hop will be B_1 if rw decides to jump to a heterogeneous node.

We repeat multiple random walks over each node in the heterogeneous graph, thereby obtaining a large number of node sequences. A skip-gram model will be trained by treating the generated node sequences as input.

4.2.2. Heterogeneous Network Embedding

A skip-gram model trains parameters by maximizing the co-occurance probability that two nodes appear in a window of length k in a random walk

[9]. During the process of model training, each input sample will cause the entire weight matrix to be updated. Therefore, the traning process leads to high time cost when the number of nodes or the dimension of embedding vecotrs is very large. In order to solve this problem, the skip-gram model introduces the negative sampling technique. It allows each training sample to update a small part of the weight matrix, which significantly reduces the computation cost of gradient descent. Let $f(\cdot)$ denote the mapping function from nodes to embedding vectors. The objective function of the skip-gram model is:

$$log\sigma(f(u) \cdot f(v)) + \sum_{m=1}^{M} \mathbb{E}_{x^m \sim P(x)}[log\sigma(-f(x^m) \cdot f(u))] \tag{6}$$

where $\sigma(\cdot)$ is the sigmoid function, i.e. $\sigma(\cdot) = \frac{1}{1+e^{-x}}$. $P(x)$ is a predefined distribution from which a negative node x^m is drawn M times [36].

4.2.3. Inter-community Link Recommendation

Based on the embedding vectors of nodes, we need to calculate the similarities of pairs of nodes and prioritize the inter-community links as recommendation results. There are various ways of calculating scores, such as using different distance metrics and performing subsequent computation by learning edge features from node embeddings. In this paper, we utilize the cosine similarity between two nodes. The cosine similarity is computed as:

$$cossim(u, v) = \frac{f(u) \cdot f(v)}{|f(u)||f(v)|} = \frac{\sum_{i=1}^{n} f_i(u) \times f_i(v)}{\sqrt{\sum_{i=1}^{n} (f_i(u))^2} \times \sqrt{\sum_{i=1}^{n} (f_i(v))^2}} \tag{7}$$

where $cossim(u, v)$ is in $[-1, 1]$. Considering that intra-community links are denser than inter-community links in HeINs, we still need to introduce the linear function in Chapter 4.1.4 to emphasize nodes across communities.

4.2.4. Complete Algorithm

Algorithm 6 presents the complete algorithm of link recommendation across communities in HeINs. RW records all node sequences sampled from multiple random walk procedures. We use *similarities* to store the cosine similarities of pairs of nodes, and it is initialized to be empty. Firstly, we perform the random walk n times over each node v. In line 5, according to the sampled node sequences, $NetworkEmbedding$ trains a skip-gram model and outputs the mapping function $f(\cdot)$ by stochastic gradient descent. The low dimensional representations of nodes preserve the structure properties and the community features. Next, we calculate the cosine similarities of n-odes which are not connected to each other in lines 6-9. Finally, the function *EnhanceICLinks* is used to strengthen the scores of inter-community pairs of nodes and save the recommendation results into *scores*.

Algorithm 6: JSBD$(G, n, l_{max}, \alpha, m, \beta, k, d, w)$

```
Input: $G = (V, E, \mathcal{R}, \mathcal{T})$: heterogeneous information network,
$n$: number of random walk iterations,
$l_{max}$: maximum length of a random walk,
$\alpha$: initial stay parameter,
$m$: number of memoried types,
$\beta$: original bias parameter,
$k$: upper bound of exploration range,
$d$: dimension of vectors,
$w$: window size
Output: $scores$: results of inter-community link recommendation
1  $RW$, $similarities \leftarrow$ empty lists
2  for $i \leftarrow 1$ to $n$ do
3      foreach $v \in V$ do
4          $RW$.add(JSBDRandomWalk($G$, $v$, $l_{max}$, $\alpha$, $m$, $\beta$, $k$))
5  $f$ = NetworkEmbedding($RW$, $w$, $d$)
6  foreach $v \in V$ do
7      foreach $u \in V$ do
8          if $(u, v) \notin E$ and $(v, u) \notin E$ then
9              $similarities$.add($cossim(f(u), f(v))$)
10 $scores$ = EnhanceICLinks($similarities$)
11 return $scores$
```

5. Evaluation

5.1. Experiments on HoINs

In order to evaluate the accuracy and efficiency of the proposed method *CNLH*, we conduct experiments on homogeneous real-world and synthetic datasets. Moreover, we present a case study to examine the recommendation quality of our method in practical scenarios.

5.1.1. Dataset Description

Due to the diversity of community detection metrics, we empirically evaluate our method on three real-world datasets that have ground-truth communities.

- **Eu-core data** [54]: It consists of email communications between members of a large European research institution. An edge (u, v) exists in the network if person u sent person v at least one email. Each community represents a department at the institution, and each individual belongs to exactly one community.
- **DBLP data** [55]: It is a co-authorship network in computer science. In DBLP, nodes stand for authors, and edges represent the co-authorships between them. The publication venues, such as journals or conferences, are considered as forming communities of nodes.
- **Amazon data** [55]: In this dataset, nodes represent products, and edges denote a co-purchase of two products. Nodes are organized into communities according to the product categories provided by Amazon.

The statistics of the three real datasets are given in Table 1.

Table 1: Statistics of homogeneous real-world datasets

Dataset	|V|	|E|	|Communities|
Eu-core	1005	25571	42
DBLP	317080	1049866	13477
Amazon	334863	925872	75149

We also generate synthetic datasets by the LFR benchmark graphs [56]. It extends the GN benchmark [57] by introducing features of real netowrks and assumes that the degree and community size distributions follow power laws. It can generate benchmarks of variable sizes and average degrees, and has been used to generate synthetic datasets in many community-detection studies [58, 59]. Given the number of nodes, the average degree and the mixing parameter, [56] generates an undirected network that meets the criteria, where the mixing parameter can adjust the proportion of links across and within communities. The mixing parameter is a key factor that affects accuracy. Therefore, fixing other parameters, we tune the mixing parameter and generate three synthetic datasets. We fix the number of nodes to 10000, the average degree to 5 and the maximum degree to 20. The mixing parameter is tested with 0.2, 0.4 and 0.6.

5.1.2. Experimental Setting

In this section, we introduce the comparison methods, the detailed experimental setups, and the metrics used to evaluate the performance of the methods.

1) Comparison Methods: Due to the high time cost of probability-based, matrix-factorization based, and classifier-based methods, we compare our methods with path-based and network-embedding based methods. More information about the methods being compared is described as follows:

- **Local Random Walk (LRW)** [17]: It is a local path-based method based on random walk. It computes the similarities according to the probability that one node reaches another after a t-step random walk procedure. It reduces the computational complexity by limiting t.
- **SimRank** [18]: It assumes two nodes are similar when they are related to similar nodes. A distance r is used to prune the branches out of the computation range in practice.
- **Node2vec** [5]: It is a network-embedding based approach that can learn continuous feature representations of nodes, which organize nodes based on their network roles and communities they belong to.
- **SEAL** [35] By extracting a subgraph around each target link, it develops a γ-decaying heuristic theory and proves that all heuristics unified in a single framework can be well approximated by local subgraphs. Thereby, a graph neural network is used to learn the heuristics from subgraphs.

2) Parameter Setup: The parameters are set as follows, according to the settings in previous work. The t of LRW is 4 on real datasets, and is 6 on synthetic datasets. The r of SimRank is set to 5. The parameters of Node2vec are set to $d = 128$, $r = 10$, $l = 80$, and $k = 10$. To improve the proportion of DFS, q is 0.5, and p is 1. The *hop* of SEAL is set to 1. It is not necessary to train SEAL on all links in the training data especially when the observed links are numerous. Therefore, the *max-train-num* is 100,000. Other parameters of SEAL are set to default. All the results of

Table 2: Parameters on different datasets, where avg.s represents the average similarity.

Datasets	l	iter	α	$[\epsilon_1, \epsilon_2]$	h	avg.s	δ
Eu-core	20	1000	0.8	[0.3, 0.5]	3	0.15	0.04
DBLP	20	50000	0.8	[0.3, 0.5]	4	0.32	0.08
Amazon	20	50000	0.8	[0.3, 0.5]	5	0.21	0.05
bm_0.2	20	1000	0.8	[0.3, 0.5]	6	0.04	0.01
bm_0.4	20	1000	0.8	[0.3, 0.5]	6	0.02	0.005
bm_0.6	20	1000	0.8	[0.3, 0.5]	6	0.008	0.001

competing methods are transformed with the same function described in Section 4.1.4 for a fair comparison. The parameters of our method include the random walk step l, the number of random walk iterations $iter$, the bias parameter of random walk α, the interval of determining core nodes $[\epsilon_1, \epsilon_2]$, the upper bound of limited-hop traversal h, and the threshold to enhance inter-community links δ. Specific parameter values used in our experiments are presented in Table 2. The first four parameters determine the core node set. Fixing α and $[\epsilon_1, \epsilon_2]$, the core node set is steady when $l \cdot iter$ is near or greater than $|E|$. h is related to the average degree and the average number of nodes in each community. Therefore, we adjust $iter$ and set h to 3, 4, 5, and 6 according to the number of edges and the scales of datasets, respectively. For instance, due to the small $|E|$, $iter$ is set to 1,000 in Eu-core and synthetic datasets. h is set to 6 in three synthetic datasets because when the mixing parameter is small, a larger h provides a better exploration effect. To obtain an optimal δ, we test some values lower than the average similarity of each dataset and select a roughly unified value $0.25 * avg.s$ as a reference. We also study the effects of tuning different parameters in later experiments and provide some strategies related to parameter selections along with the analysis.

In contrast to the traditional link prediction problem, we compose the labeled testing set of edges as follows: to obtain positive samples, we remove 10% of E_{ac} chosen from the network, and to obtain negative samples, we remove an equal number of edges from E_{ic}. We remove edges randomly while ensuring the residual network is connected. We use the remaining 90% of E_{ac} as well as the remaining E_{ic} to construct the training data. It is worth noting that some nodes belong to more than one community in real datasets. Due to the difficulty in deciding the labels of related links, we do not sample these edges in our experiments.

All algorithms in the paper are implemented in Java, including LRW and SimRank. For Node2vec and SEAL, we use the original Python implementation provided by the authors [60, 61], where an optimized library word2vec and a PyTorch implementation of DGCNN (Deep Graph Convolution Neural Network) [62] are applied, respectively. All experiments are performed on a machine with an Intel Core i7 3.40-GHz processor and a 16-GB memory. Since our CNLH algorithm is highly parallelizable, we utilize the multi-threading technique to improve efficiency. Each thread is responsible for processing a part of core nodes. The code is publicly available at `https://github.com/ShawUz/LPAC`.

3) Evaluation Metrics: Methods can output the similarity scores of potential inter-community and intra-community links among nodes in the target network. In this comparison, we mainly study two aspects: (1) accuracy: Area Under Curve (AUC) [63] is used to test accuracy. A higher AUC score represents higher accuracy. (2) efficiency (time consumption).

5.1.3. Experimental Results

In addition to the three competing methods, we also compare our method (CNLH) with neighbor-based methods AA [12], HDI [64], HPI [64], JI [11], PA [14], and RA [13] on the Eu-core dataset. The results are shown in Table 3. However, since the neighbor-based methods only consider distance-two pairs of nodes, limiting the ability to reach nodes far away, we omit these methods in the following experiments.

Table 3: Performance comparisons with neighbor-based methods on Eu-core dataset.

Method	CNLH	AA	HDI	HPI	JI	PA	RA
AUC	0.723	0.592	0.698	0.687	0.692	0.415	0.667

Next, we compare our method with three competing methods over real datasets. Figures 8(a) and 8(b) show the results. It takes more than 24 hours for LRW and SimRank to run on DBLP and Amazon datasets, so we terminate them before they finish. Although path-based methods show competitive accuracy results, the time consumption makes them impractical for massive datasets. In contrast, our method performs better on larger datasets. The run time of our method is about half of that of Node2vec on Eu-core and DBLP datasets, and CNLH is 17 times faster than Node2vec on the Amazon dataset. Moreover, it is one to two orders of magnitude

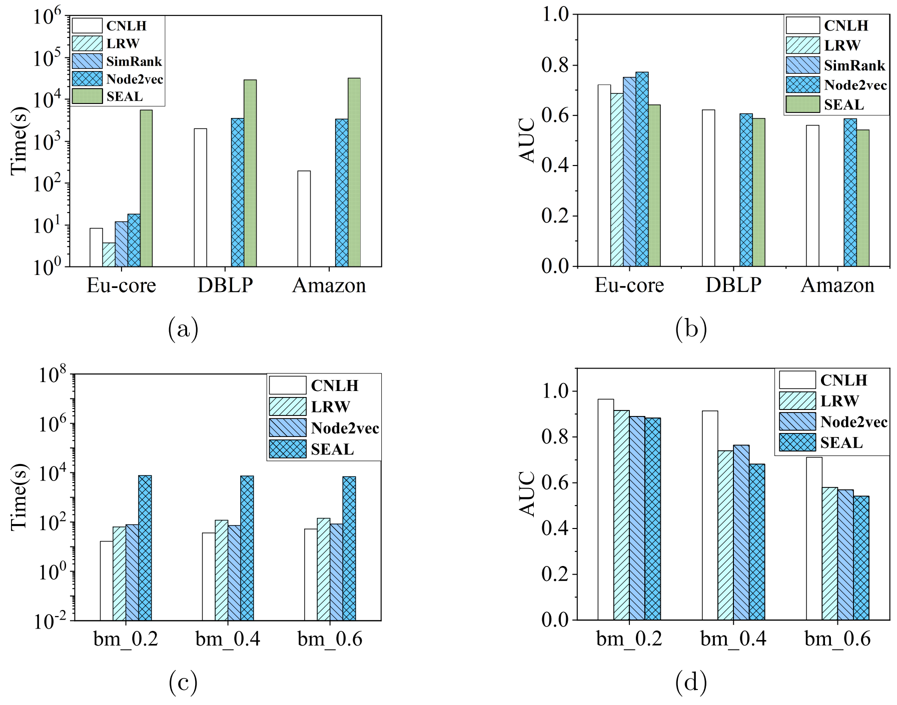


Figure 8: Performance comparisons on real-world and synthetic datasets.

faster than SEAL. In terms of accuracy, CNLH is comparable to competing methods on Eu-core and Amazon, and shows a better accuracy on DBLP.

Although the real-world datasets include the so-called ground-truth communities, they may not reflect the absolutely accurate community situations in reality. For instance, the Amazon dataset has removed the ground-truth communities whose number of nodes is less than 3, but the average size of communities is still only 4.4, and there are multiple overlaps between communities. The boundaries between communities are unclear. Therefore, we generate a few synthetic datasets to evaluate our method. The results are shown in Figures 8(c) and 8(d). Our method is significantly faster than other methods. The results of SimRank are omitted since it runs too long. The accuracy decreases as the mixing parameter increases due to the increasing difficulty of distinguishing communities. Our proposed method shows higher accuracy and less time cost on all tested synthetic datasets.

Next, we test the effects of varying different parameters — we provide strategies for choosing suitable parameter values, making our method easy

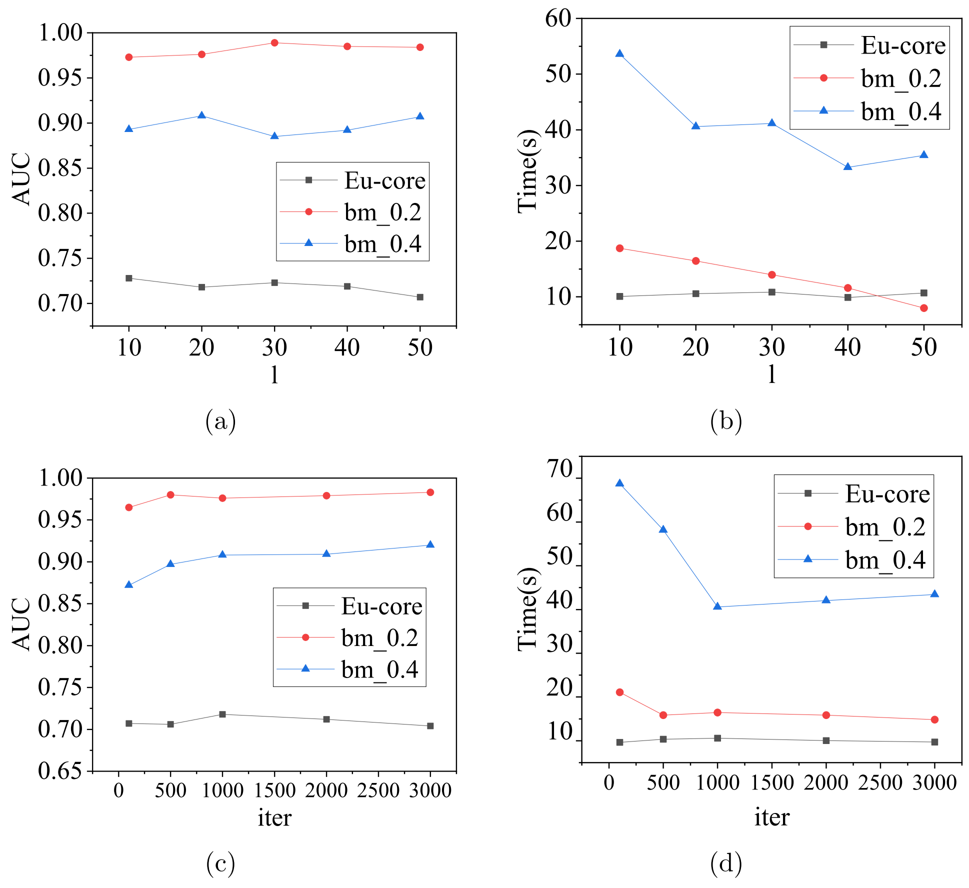
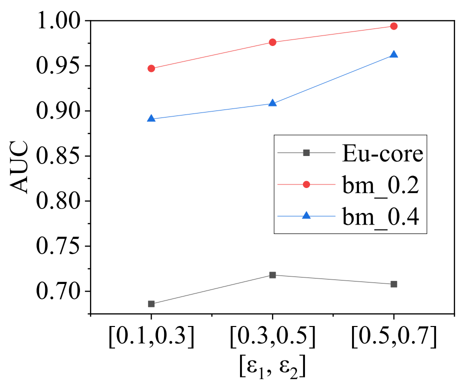
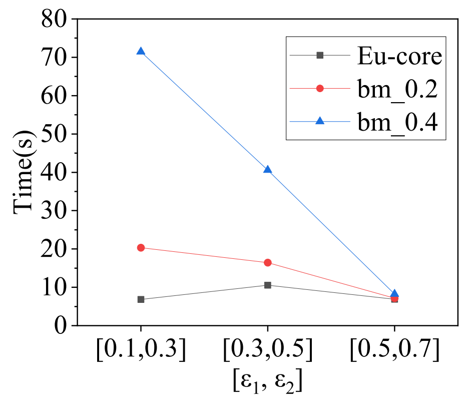


Figure 9: The effects of parameters l and $iter$.

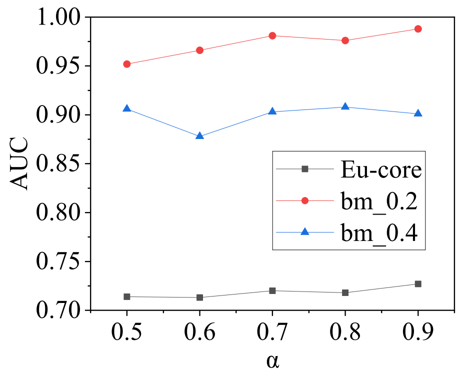
to use in practice. We keep other parameters consistent with those in Table 2. Figures 9 and 10 show the results. Parameters l, $iter$, and $[\epsilon_1, \epsilon_2]$ mainly affect the construction of the core node set. Accuracy is relatively stable on tested datasets when tuning l. As $iter$ increases, accuracy increases at the beginning and remains steady. For time costs, they present a downward trend and eventually reach stability. This indicates that insufficient sampling leads to an increase in the number of core nodes and fluctuant quality of the core node set. The core node set gradually approaches a steady state as l and $iter$ increase, and $iter$ presents a larger effect on accuracy. When $iter$ is sufficiently large, there is little fluctuation on the AUC curve and the time cost curve. We recommend that l lies in [10, 50], and $l \cdot iter \geq |E|$ should be satisfied, where l can be adjusted according to the scale of the network.



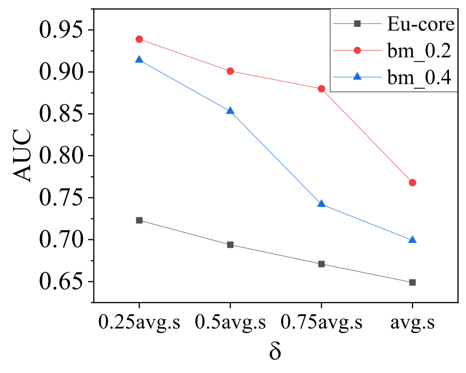
(a)



(b)



(c)



(d)

Figure 10: The effects of parameters $[\epsilon_1, \epsilon_2]$, α, and δ.

Setting high filtering intervals results in significantly lower time cost, since high filtering intervals imply fewer core nodes. The interval can be adjusted on the basis of specific requirements. For example, if we expect to obtain more core nodes, the interval should be extended to the lower bound and the upper bound. The interval should be higher when we only need to focus on more important core nodes. It is worth noting that a very small lower bound will cause numerous overlaps of similarity calculation. Therefore, setting the lower bound too small is not recommended. The interval can also be determined according to the distribution of the centralities of nodes in specific networks. The results of varying α show that a larger α results in a higher accuracy, indicating that a bias toward DFS is beneficial during the sampling process. This is because DFS prefers to capture the importance of a node in connectivity, helping nodes to explore further and generating inter-community links. Generally, α should be larger than 0.5, and in order to increase the proportion of DFS, an α in [0.7, 0.9] is recommended. Figure 10(d) shows the effect of δ on accuracy. The accuracy decreases as δ increases since the number of links across communities is less than the number of links within communities. Therefore, δ should be smaller than $avg.s$, and a relatively small δ provides better accuracy. We recommend a δ close to $0.25 * avg.s$. Consequently, parameters guiding our computation procedure can be adjusted according to the scale and the characteristics of the datasets, giving us control over the recommendation range.

5.1.4. Case Study

We have shown in evaluations that Node2vec has comparable accuracy with ours in terms of the AUC metric. In addition to this strict numerical evaluation criteria, we would like to compare how they would perform in practical scenarios. We assume that each method recommends links with top scores.

Consider the Amazon dataset, which is a co-purchasing network. In this dataset, each node represents a product, and the product categories are used as community labels, such as <Subjects | Computers & Internet | Programming | Java | Genral>. Through observing the top-5% recommendations, we find that our link recommendation method is able to recommend products far enough in topology, while maintaining the relevance between the recommended products. It is exactly what we expect to achieve by the limited-hop traversal. Since Node2vec is capable of calculating the similarities between all pairs of nodes, some pairs that are particularly far away may be recommended as well, resulting in unreasonable or useless recommendations. For instance, there is a book that belongs to <Subjects | Health, Mind & Body | Diets & Weight Loss | Diets | General>, and our recommendation is a book in <Subjects | Health, Mind & Body | Personal Health | Women's Health | Genaral>. By contrast, node2vec treats a book in <Subjects | Cooking, Food & Wine | General> and another in <Subjects | Literature & Fiction | General | Contemporary> as a recommendation. Therefore, the recommendations of our method are more reasonable than Node2vec's in this real application scenario while ensuring a sufficiently far distance in topology at the same time.

5.2. Experiments on HeINs

In this section, we demonstrate the effectiveness of JSBD by conducting experimental evaluations compared to the state-of-art network embedding methods. We first introduce four heterogeneous datasets, the comparative methods, and the setup of parameters. Then, we show the experimental results of JSBD and other baselines. Finally, we study the performance of our method in practical scenarios.

5.2.1. Datasets

We adopt four real-world heterogeneous datasets to evaluate the performance of our method and the baseline methods. The statistics of the datasets are summarized in Table 4.

Table 4: Statistics of heterogeneous real-world datasets

Datasets	Relations(A-B)	—A—	—B—	—(A-B)—
Magazine	User-Magazine	348	157	2,375
	Magazine-Brand	157	61	154
	Magazine-Magazine	157	157	1,012
Movies	Actor-Movie	10,789	7,332	24,043
	Actor-Actor	10,789	10,789	48,088
	Movie-Director	7,332	1,741	6,156
	Movie-Compose	7,332	1,483	4,155
	Movie-Movie	7,332	7,332	6,596
Grocery	User-Grocery	127,496	41,320	1,143,860
	Grocery-Brand	41,320	8,869	40,895
	Grocery-Grocery	41,320	41,320	810,379
Douban	User-Movie	13,367	12,677	1,068,278
	User-User	2,440	2,294	4,085
	Movie-Director	10,179	2,449	11,276
	Movie-Actor	11,718	6,311	33,587
	Movie-Type	12,678	38	27,688

- **Magazine data** [65]: This dataset includes reviews and metadata of magazine products from Amazon, and it has been reduced to extract the dense subsets, such that all users and magazines have at least 5 reviews. As there is no co-purchase interaction in the original dataset, we enrich this graph by adding the magazine-magazine edges according to the also_bought field of metadata. The ground truth communities are divided according to magazine categories.

- **Movies data** [37]: This is a dataset that contains information about movies, actors, directors, and composers. Each movie has one or more genre labels, including action, horror, adventure, scifi, and crime, which are used as ground truth community labels. Nodes are considered to belong to the same community if they have more than one common genre.

- **Grocery data** [65]: Grocery is from the reviews of groceries and gourmet foods sold by Amazon. Similar to Magazine, it is the 5-core version of the complete dataset, and we add homogeneous edges between products if they were co-purchased together.

- **Douban data** [66]: It is a dataset of movie domain from the rating platform Douban. It contains social relations of users and properties of movies. For this dataset, multiple groups naturally divide users into different clusters. Therefore, the target type of link recommendation across communities in Douban is user.

There are some nodes that belong to more than one community in four datasets. We treat them as nodes that belong to different communities when the intersections of their labels are empty. Note that the number of inter-community links between the target-type nodes is small, we remove 50% chosen edges from them randomly as positive samples of testing set, and we sample an equal number of edges from intra-community links to generate negative samples. The remaining edges are used as the training data.

5.2.2. Comparison Methods and Parameter Settings

We compare the proposed algorithm JSBD with the following network embedding methods, among which DeepWalk, LINE, and Node2vec are suitable for general network embedding.

- **DeepWalk** [34]: It constructs node sequences by random walks and applies a skip-gram model to learn the network embedding. We set the number of iterations starting from each node $\gamma = 10$, the length of random walk $t = 100$, and the window size of the skip-gram model $w = 10$ according to the original settings.

- **LINE** [38]: It preserves the first-order and second-order proximities by defining a novel objective function. It adopts an efficient edge sampling method to reduce the time cost and can be extended to large-scale networks easily. Except the dimension of vectors, parameters are set to default.

- **Node2vec** [5]: This method designs a biased random walk procedure to explore diverse neighbors and capture the community features of nodes. We set the return parameter $p = 1$ and in-out parameter $q = 0.5$ according to the original settings.

- **Hin2vec** [39]: Hin2vec exploits different types of interactions and performs multipe prediction tasks to jointly learn vectors of nodes and meta-paths. We set the maximum length of meta-paths to 3, and other parameters are the same as in DeepWalk.

- **JUST** [9]: It adopts random walks with stay and jump strategies to balance the distribution of different types in HeINs. The initial stay parameter is set to 0.5. We expect to confirm the effect of the biased strategy in JSBD by comparing with this method.

- **Metapath2vec** [36]: It applies predefined meta-paths to guide random walks and then leverages the skip-gram model to learn node representations. In our experiments, we define different meta-paths for each dataset and show the best one. We construct meta-paths as follows: Magazine (magazine-user-magazine, magazine-brand-magazine), Movies (actor-movie-director-movie-actor, actor-movie-composer-movie-actor), Grocery (grocery-user-grocery, grocery-brand-grocery), and Douban (user-movie-user, user-movie-director-movie-user). The parameter settings are the same as in DeepWalk.

- **PTE** [40]: It learns text embedding by labeled data and unlabeled data. We restrain it as an unsupervised model and construct bipartite graphs for each dataset: Magazine (magazine-user, magazine-brand), Movies (movie-actor, movie-director, movie-composer), Grocery (grocery-user, grocery-brand), and Douban (user-movie, user-user). We set the same parameters as in DeepWalk.

For our algorithm, we set the upper bound of exploration range $k = 6$, the original bias parameter $\beta = 9$, the maximum length of a random walk $l_{max} = 100$, the number of random walk iterations $n = 10$, the original stay parameter $\alpha = 0.5$, and the window size of the skip-gram model $w = 10$. The number of memorized types m is set to 2 for Movies and Douban, and it is set to 1 for Magazine and Grocery. For a fair comparsion, the embedding dimension d of all models is set to 128.

5.2.3. Experimental Results

We transform the similarities of all methods into final scores by the same linear function for a fair comparison, where the threshold δ is tuned in $[0.3 * avg.s, 0.7 * avgs]$, and we report the best result. Figure 11 shows the results of the four heterogeneous datasets.

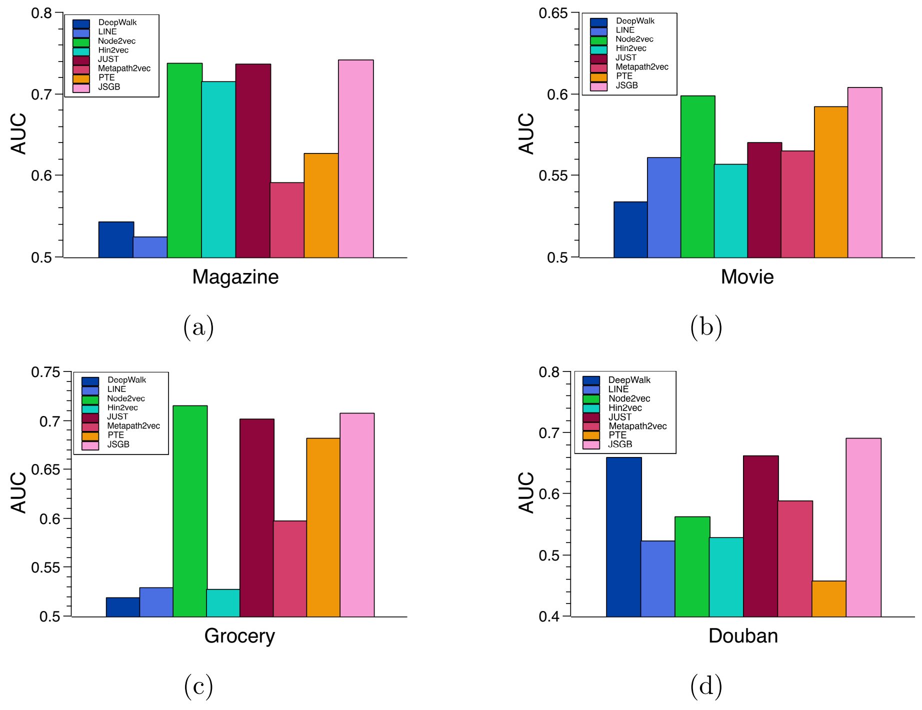


Figure 11: Performance comparisons on heterogeneous datasets.

Figures 11 (a), (b), (c), and (d) are the comparison results in terms of the AUC metric of our method JSDB and other embedding methods on Magazine, Movies, Grocery, and Douban datasets, respectively. It is clear that our proposed method outperforms all competing methods on Magazine, Movies, and Douban. JSBD only lags behind the best performing method 1.12% on Grocery. These results indicate that our method is capable of accurately preserving the topology structure and the community feature of nodes in the embedding vectors.

5.2.4. Case Study

Apart from the accuracy metric, it is also important that a model can provide suitable and userful recommendations in real applications. Compared with other baselines, which do not intentionally consider the community characteristic, Node2vec shows a better performance. Therefore, we focus on the performance of JSBD and Node2vec on specific recommendations.

Table 5: Part of recommendation results of Grocery

Purchased items	Recommended items
Snack Foods, Jerky | Dried Meats	JSBD: Meat & Seafood | Bacon Node2vec: Beverages | Coffee | Tea & Cocoa | Tea
Candy & Chocolate | Brittle | Caramel & Toffee, Brittle	JSBD: Beverages | Coffee | Tea & Cocoa | Tea | Iced Tea Node2vec: Cooking & Baking | Nuts & Seeds | Pistachios
Pasta & Noodles | Noodles | Rice	JSBD: Cooking & Baking | Flours & Meals | Wheat Flours & Meals Node2vec: Snack Foods | Salasa | Dips & Spreads
Olives | Pickles & Relishes | Relishes | Vegetable Relishes	JSBD: Herbs | Spices & Seasonings | Single Herbs & Spices | Allspicse Node2vec: Candy & Chocolate | Candy & Chocolate Bars
Condiments & Salad Dressings | Salad Dressings | Ranch	JSBD: Sauces | Gravies & Marinades | Sauces | Asian | Sweet & Sour Sauce Node2vec: Snack Foods | Crackers | Assortments & Samplers
Candy & Chocolate | Candy & Chocolate Bars	JSBD: Beverages | Coffee | Tea & Cocoa | Tea Node2vec: Breakfast Foods | Breakfast & Cereal Bars | Cereal

We use the product recommendation on Grocery as a case study. Each product of Grocery is assigned a community label, such as ¡Dairy | Cheese & Eggs | Milk Substitutes | Coconut Milk¿. We pick top-5% recommendations from the results and report some examples in Table 5. The recommendations of JSBD not only present higher correlation with purchased products than those of Node2vec, but also ensure a distance in the topological structure. This is consistent with the purpose that we propose the problem of link recommendation across communities. Consequently, the superiority of our method becomes more significant in practical scenarios.

6. Conclusions

Link recommendation is useful in many scenarios, and is often solved by link prediction methods. Current link prediction methods dominantly assign a high score for a pair of nodes within a short-distance range, leading to the lack of diversified recommendations. We propose to study a novel problem called *link recommendation across communities* in this paper. It expects to find links potentially valuable across communities (not necessarily very far in terms of path length—due to the small world phenomenon) that are often overlooked by conventional link recommendation methods. The prioritization of inter-community link prediction naturally extends link prediction and increases the diversity of recommendation systems. To solve this problem, we present a solution based on biased random walks and limited-hop traversals to compute similarities in HoINs. For HeINs, we propose a network embedding model to capture the network structure and community feature. We compare our method with neighbor-based, path-based, and network-embedding based methods. The comprehensive experiments on real-world and synthetic datasets demonstrate that our algorithms provide a trade-off in balancing computational efficiency and predictive accuracy, while showing better performance in practical scenarios.

References

[1] C. Lei, J. Ruan, A novel link prediction algorithm for reconstructing protein-protein interaction networks by topological similarity, Bioinformatics 29 (3) (2012) 355–364.

[2] D. Liben-Nowell, J. Kleinberg, The link-prediction problem for social networks, Journal of the American Society for Information Science and Technology 58 (7) (2007) 1019–1031.

[3] H. Chen, X. Li, Z. Huang, Link prediction approach to collaborative filtering, in: Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries, 2005, pp. 141–142.

[4] J. C. Cubero, A Survey of Link Prediction in Complex Networks, ACM, 2016.

[5] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.

[6] M. S. Granovetter, The strength of weak ties, American Journal of Sociology 78 (6) (1973) 1360–1380.

[7] A. Biswas, B. Biswas, Defining quality metrics for graph clustering evaluation, Expert Systems with Applications 71 (C) (2016) 1–17.

[8] J. Kleinberg, The small-world phenomenon: An algorithmic perspective, in: ACM Symposium on Theory of Computing, 2000.

[9] R. Hussein, D. Yang, P. Cudré-Mauroux, Are meta-paths necessary?: Revisiting heterogeneous graph embeddings, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, ACM, 2018, pp. 437–446.

[10] L. Lü, T. Zhou, Link prediction in complex networks: A survey, Physica A: Statistical Mechanics and its Applications 390 (6) (2011) 1150–1170.

[11] P. Jaccard, Étude comparative de la distribution florale dans une portion des alpes et des jura, Bulletin de la Societe Vaudoise des Science Naturelles 37 (142) (1901) 547–579.

[12] L. A. Adamic, E. Adar, Friends and neighbors on the web, Social Networks 25 (3) (2003) 211–230.

[13] T. Zhou, L. Lü, Y.-C. Zhang, Predicting missing links via local information, The European Physical Journal B 71 (4) 623–630.

[14] M. Mitzenmacher, A brief history of generative models for power law and lognormal distributions, Internet Mathematics 1 (2) (2004) 226–251.

[15] L. Katz, A new status index derived from sociometric analysis, Psychometrika 18 (1) (1953) 39–43.

[16] L. Lü, C.-H. Jin, T. Zhou, Similarity index based on local paths for link prediction of complex networks, Physical Reivew E 80 (4) (2009) 046122.

[17] W. Liu, L. Lü, Link prediction based on local random walk, Europhysics Letters 89 (5) (2010) 58007–58012(6).

[18] G. Jeh, J. Widom, Simrank: a measure of structural-context similarity, in: Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, pp. 538–543.

[19] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, in: Proceedings of the International Conference on World Wide Web, Vol. 30, 1998, pp. 107–117.

[20] J. Neville, D. Jensen, Relational dependency networks, Journal of Machine Learning Research 8 (2) (2001) 653–692.

[21] E. M. Airoldi, D. M. Blei, S. E. Fienberg, E. P. Xing, Mixed membership stochastic block models, in: Proceedings of the International Conference on Neural Information Processing Systems, 2008, pp. 33–40.

[22] A. Clauset, C. Moore, M. E. Newman, Hierarchical sturcture and the prediction of missing links in networks, Nature 453 (7191) (2008) 98.

[23] Z. Huang, Link prediction based on graph topology : the predictive value of generalized clustering coefficient, Social Science Electronic Publishing (2006).

[24] A. K. Menon, C. Elkan, Link prediction via matrix factorization, Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (1) (2011) 437–452.

[25] L. Duan, S. Ma, C. Aggarwal, T. Ma, J. Huai, An ensemble approach to link prediction, IEEE Transactions on Knowledge and Data Engineering 29 (11) (2017) 2402–2416.

[26] M. Al Hasan, V. Chaoji, S. Salem, M. Zaki, Link prediction using supervised learning, Proceedings of SDM Workshop on Link Analysis Coutertterrorism & Security 30 (9) (2006) 798–805.

[27] R. N. Lichtenwalter, J. T. Lussier, N. V. Chawla, New perspectives and methods in link prediction, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 243–252.

[28] H. R. De Sá, R. B. Prudêncio, Supervised link prediction in weighted networks, in: Proceedings of the International Joint Conference on Neural Networks, 2011, pp. 2281–2288.

[29] H. Kashima, T. Kato, Y. Yamanishi, M. Sugiyama, K. Tsuda, Link propagation: a fast semi-supervised learning algorithm for link prediction, in: Proceedings of the 2009 SIAM Internation Conference on Data Mining, 2009, pp. 1100–1111.

[30] R. Raymond, H. Kashima, Fast and scalable algorithms for semi-supervised link prediction on static and dynamic graphs, in: Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 2010, pp. 131–147.

[31] Z. Zeng, K.-J. Chen, S. Zhang, H. Zhang, A link prediction approach using semi-supervised learning in dynamic networks, in: Proceedings of the Internation Conferennce on Advanced Computational Intelligence, 2013, pp. 276–280.

[32] Y.-L. Chen, M.-S. Chen, S. Y. Philip, Ensemble of diverse sparsifications for link prediction in large-scale networks, in: Proceedings of the IEEE International Conference on Data Mining, 2015, pp. 51–60.

[33] M. Ou, P. Cui, J. Pei, Z. Zhang, W. Zhu, Asymmetric transitivity preserving graph embedding, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 1105–1114.

[34] B. Perozzi, R. Alrfou, S. Skiena, Deepwalk: Online learning of social representations, in: Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, 2014.

[35] M. Zhang, Y. Chen, Link prediction based on graph neural networks, in: Advances in Neural Information Processing Systems, 2018, pp. 5165–5175.

[36] Y. Dong, N. V. Chawla, A. Swami, metapath2vec: Scalable representation learning for heterogeneous networks, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017.

[37] Z. Huang, N. Mamoulis, Heterogeneous information network embedding for meta path based proximity, CoRR abs/1701.05291 (2017).

[38] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large-scale information network embedding, in: Proceedings of the 24th international conference on world wide web, International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[39] Y. Fu, W.-C. Lee, Z. Lei, Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, ACM, 2017, pp. 1797–1806.

[40] J. Tang, M. Qu, Q. Mei, Pte: Predictive text embedding through large-scale heterogeneous text networks, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 1165–1174.

[41] H. Chen, H. Yin, W. Wang, H. Wang, Q. V. H. Nguyen, X. Li, Pme:projected metric embedding on heterogeneous networks for link prediction, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 1177–1186.

[42] W.-Y. Chen, D. Zhang, E. Y. Chang, Combinational collaborative filtering for personalized community recommendation, in: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2008, pp. 115–123.

[43] A. Sharma, B. Yan, Pairwise learning in recommendation: experiments with community recommendation on linkedin, in: Proceedings of the 7th ACM Conference on Recommender Systems, ACM, 2013, pp. 193–200.

[44] X. Han, L. Wang, R. Farahbakhsh, Á. Cuevas, R. Cuevas, N. Crespi, L. He, Csd: A multi-user similarity metric for community recommendation in online social networks, Expert Systems with Applications 53 (2016) 14–26.

[45] R. Kumar, J. Novak, A. Tomkins, Structure and evolution of online social networks, in: Link mining: models, algorithms, and applications, Springer, 2010, pp. 337–357.

[46] R. Diestel, Graph theory, Springer Publishing Company, Incorporated, 2018.

[47] L. C. Freeman, A set of measures of centrality based on betweenness, Sociometry (1977) 35–41.

[48] P. Csermely, A. London, L. Y. Wu, B. Uzzi, Structure and dynamics of core/periphery networks, Journal of Complex Networks 1 (2) (2013) 93–123.

[49] https://en.wikipedia.org/wiki/Core-periphery_structure.

[50] F. Della Rossa, F. Dercole, C. Piccardi, Profiling core-periphery network structure by random walkers, Scientific reports 3 (2013) 1467.

[51] F. L. Bauer, The Plankalkl of Konrad Zuse: a forerunner of today's programming languages, 1972.

[52] T. T. Cormen, C. E. Leiserson, R. L. Rivest, Introduction to algorithms, Resonance 1 (9) (2003) 14–24.

[53] G. Salton, M. J. McGill, Introduction to Modern Information Retrieval, MuGraw-Hill, Auckland (1986).

[54] J. Leskovec, J. Kleinberg, C. Faloutsos, Graph evolution:densification and shrinking diameters, Acm Transactions on Knowledge Discovery from Data 1 (1) (2007) 2.

[55] J. Yang, J. Leskovec, Defining and evaluating network communities based on ground-truth, Knowledge and Information Systems 42 (1) (2015) 181–213.

[56] A. Lancichinetti, S. Fortunato, F. Radicchi, Benchmark graphs for testing community detection algorithms, Physical Review E 78 (4 Pt 2) (2008) 046110.

[57] M. Girvan, M. E. J. Newman, Community structure in social and biological networks 99 (12) (2002) 7821–7826. doi:10.1073/pnas.122653799.

[58] F. Folino, C. Pizzuti, An evolutionary multiobjective approach for community discovery in dynamic networks, IEEE Transactions on Knowledge & Data Engineering 26 (8) (2014) 1838–1852.

[59] J. Huang, H. Sun, J. Han, H. Deng, Y. Sun, Y. Liu, Shrink: A structural clustering algorithm for detecting hierarchical communities in networks, in: Acm International Conference on Information & Knowledge Management, 2010.

[60] https://github.com/aditya-grover/node2vec.

[61] https://github.com/muhanzhang/SEAL.

[62] https://github.com/muhanzhang/DGCNN.

[63] T. Fawcett, An introduction to roc analysis, Pattern recognition letters 27 (8) (2006) 861–874.

[64] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, A.-L. Barabási, Hierarchical organization of modularity in metabolic networks, Science 297 (5586) (2002) 1551–1555.

[65] http://jmcauley.ucsd.edu/data/amazon/.

[66] http://movie.douban.co.